WAIT: Workshop on the Advances in Information Technology, Budapest, 2015

# Learning 3D Object Recongition Using Graphs Based on Primitive Shapes

Marton Szemenyei,[1] Ferenc Vajda[1]

[1] Department of Control Engineering and Information Technology, Budapest University of Technology and Economics, Budapest, Hungary

**Abstract**

*Object recognition is a significant field of research with numerous practical applications.Among others, several Augmented Reality systems use some form of recognition to register real and virtual objects. Since most of these registration algorithms simply detect an artificial visual pattern, they are not feasible for uncontrolled environments. The goal of our paper is to present an object recognition algorithm that uses only natural features to match real objects to virtual ones. A tangible user interface system may use our algorithm to perform shape-based object registration, enforcing similarity of shape between real and virtual objects. Our algorithm represents the structure of a scene by creating a graph from primitive shapes, and employs a graph kernel-based SVM algorithm to classify subgraphs corresponding to certain object categories. Based on the learning classification algorithm we also developed a localization algorithm extracting the best possible subgraphs from a large scene.*

Categories and Subject Descriptors (according to ACM CCS): I.4.8 [Image Processing And Computer Vision]: Object recognition

## 1. Introduction

Augmented Reality systems have been an area of intense research in the last two decades. The goal of these systems is usually to define the next generation of user interfaces by allowing users to interact with virtual content more intuitively than before. One subfield, called Tangible Augmented Reality (TAR) combines Tangible User Interfaces (TUI) with augmented reality by introducing real-world objects into the user interface design.[1] Some systems, like Tiles[2], of MagicCup[3] use 2D objects with markers to augment virtual objects on them or perform operations with them. Several other systems use objects to project virtual images[4], or apply virtual textures[5] on them. Another interesting application, called the Virtual Round Table[6] uses random objects on a table as placeholders for virtual ones.

None of these systems concern themselves with the similarity of shape between real and virtual objects. However, objects of different form require different manipulation techniques, and provide different sensations when touched. The inconsistency of visual and haptic senses might cause neural conflict, which will greatly diminish user experience. Nonetheless, by matching real and virtual objects similar in shape this conflict can be mitigated. Moreover, a shape-based matching process enables us to create Adaptive Virtual Environments (AVE): Environments that can adapt to the specific structure of the real scene they are projected into by arranging virtual objects intelligently, so that they would fit into the real scene.

Since the shape of a scene or of certain objects is best represented as a structured object, our main focus in this paper is developing a learning algorithm for structured objects we use to describe complex shapes. Machine learning methods that work on structured objects (usually graphs) are one of the most important fields in artificial intelligence research. The application of such methods is versatile: Face, fingerprint, or general object recognition, network analysis, or molecule analysis just to mention a few.

In this paper we propose an object recognition algorithm that learns to pair real and virtual objects from instances of real objects considered a good match for the given virtual object category. The basis of recognition is the similarity between the physical properties of real and virtual objects.

However, since most physical features, such as surface texture (roughness) are difficult to measure visually, the algorithm will perform the matching based on shape only. This way the neural conflict can be mitigated. The algorithm can also be used to create AREs of varying levels of immersion.

The *first step* of the algorithm is to create a 3D reconstruction of the real scene to determine its structure. Our algorithm makes no assumptions about the spatial or visual properties of the scene, therefore no preparation step is needed before use. The *second step* is shape description. In this step we segment the scene into primitive shapes: planes, cylinders, spheres, etc. After that we compute features for each primitive shape and create a topology graph. The nodes of this graph are the primitives, while the edges encode the spatial relations of the primitives. In the *final step* of the algorithm we use a localization algorithm to find instances of virtual object categories in the real scene using a learning graph classification algorithm.

In the following section we will review the most important results of other research teams related to our algorithm. In the first part we will discuss the field of 3D shape recognition, while in the second part we will explore learning systems designed to work with graphs. In section 3, we will describe our algorithm, including the shape description, learning classification and localization methods. We will also discuss the question of selecting the optimal hyperparameters of the kernel function used by the classification algorithm. In section 4 we will describe our test methods and results, while in section 5 we summarize the outcome of our research.

## 2. Previous Work

In this chapter we will supply an overview of the results of research related to ours. In the first part of this section we will review the field of 3D shape recognition and the arising difficulties. The second part of the chapter focuses on the problem of learning on graphs, including learning graph classification and matching.

### 2.1. Shape Recognition

Shape recognition and shape matching have numerous relevant applications in the fields of computer vision and augmented reality. However, most methods recognize 2D shapes from images, and the majority of these method cannot be easily extended to 3D shapes.[7] The RANSAC[8] algorithm and the Generalized Hough Transform[9] are relevant exceptions, that work in any dimensions. Nonetheless, these algorithms require a model to match shapes against, which renders them infeasible in situations in which a reference model cannot be produced.

In situations where the reference model of the objects to be recognized is unknown, the application of machine learning algorithms to solve the shape recognition problems is recommended. Osada et. al.[7] used shape distributions to create a learning 3D shape recognition algorithm, while Golovinskiy et. al. [10] employed a shape feature-based learning algorithm to categorize objects in scans of urban environments. It is important to note, however, that in both of these cases segmentation of the objects to be recognized is needed. This might be relatively straightforward in cases, in which we have a priori information on the objects in the scene (like in urban environments), still, when no such information is available the segmentation becomes arbitrary. In these situations the quality and characteristics of segmentation may influence the results and the performance of the shape recognition algorithm greatly.

Schnabel et. al.[11] presented a third method for 3D shape recognition in point clouds. Their method begins with a segmentation step: They employ an efficient RANSAC[12] algorithm with local random sampling and candidate evaluation to break down a point cloud into primitive shapes. Their algorithm is capable of detecting five different primitives. Then, they construct a topology graph from these primitive shapes, with nodes representing the primitives themselves, while edges representing the geometric relations between primitives. The adjacency of primitive shapes is determined by using an octree representation of the point cloud.[12]

To recognize objects in this representation they use a reference graph and a brute-force graph matching algorithm. The low number of nodes in the reference graph ensures the feasibility of the brute-force algorithm. In order to decrease the number of possible matches between the reference and scene graphs they employ a number of constraints. Node constraints ensure that nodes representing different types of shapes do not get matched, while edge constraints allow the algorithm to take the relations of adjacent shapes into account. Graph-level constraints can be used to enforce relations between non-adjacent primitive shapes.[11]

### 2.2. Learning on Graphs

There are widespread applications of learning on graphs in several different areas of research, including analysis of ICT or social networks, and outlier subgraph detection. Moreover, in the field of computer vision learning on graphs can be used for face, or general object category recognition, since these objects can be represented as graphs of visual features. In this chapter we will discuss one of the learning problems related to graphs, namely graph classification.

Graph classification is a problem, where an algorithm attempts to assign a label **{0,1}** or **{0,1,...,n}** to a query graph. To achieve this, standard learning classification algorithms are often used. However, since most supervised and unsupervised classification algorithms require query and training data to be represented as a feature vector, a vectorial representation of graphs is needed.

There is one straightforward way to construct a vector from a graph. By vectorizing the adjacency matrix of a weighted graph column-wise and adding the vector of node weights (if the graph has weighted nodes as well) a feature vector can be constructed. Nonetheless, there are several problems with such a representation. First, the graph is a structured object, therefore it is invariant to the arbitrary ordering of nodes. However, the feature vector constructed using the aforementioned method will no longer be invariant to the ordering of nodes. Moreover, graphs of different sizes will produce feature vectors of different lengths, which makes comparing them difficult.

One possible solution to this problem is the spectral representation of graphs.[13] The spectral representation is computed by taking the eigenvalue or spectral decomposition (eq. 2) of a matrix representing the graph. There are a few different matrices that can be used, such as the adjacency, or the Laplacian matrix (eq. 1). By ordering the eigenvalues and the corresponding eigenvectors of the graph and constructing a single vector from them we create a representation that is partially invariant to node ordering. Naturally, this partial invariance means that an alignment step is still needed.[14] This representation is also able to describe graph of different lengths by adding dummy nodes to the smaller graphs.

$$\mathbf{L} = \mathbf{A} - \mathbf{D} \tag{1}$$

$$\mathbf{L} = \mathbf{V} diag\left(\lambda_i\right) \mathbf{V}^{\mathbf{T}} \tag{2}$$

$\mathbf{L}$ denotes the Laplacian matrix of the graph, $\mathbf{D}$ is the degree matrix, while $\mathbf{V}$ is the orthogonal and diagonal matrix containing the eigenvectors and $\lambda_i$ denotes the eigenvalues. The spectral representation then can be used with vectorial learning algorithms. White[14], for instance, mixes several different spectral representations and employs a median graph calculation methods to create a generative algorithm.

Using kernel methods provides a different solution to the problem described above. Kernel methods are learning algorithms employing kernel functions: symmetric, positive semi-definite functions that produce a similarity measure of two objects. Several learning algorithms can be used with kernel functions instead of direct representation of the data, thus, with a well-considered kernel function we can avoid vectorization of graphs altogether. Naturally, a feasible kernel function must produce a similarity score for two graphs and be invariant to the ordering of nodes and to the size of each graph.

One widely used kernel is the random walk kernel[15]. This method interprets the edge weights of the graphs as the probability of taking that edge during a random walk on the graph. The algorithm derives the similarity score of the two graphs by performing random walks on the graph simultaneously, and counting the number of walks that were the same on both graphs. It is natural, that when performing random walks on graphs with similar structure, there is a high probability that the two walks will be the same, whereas with dissimilar graphs the probability of matching walks will be low. The kernel function is shown in equation 3.

$$K = \sum_{i=1}^{N} w(i) \mathbf{e}^T \mathbf{A}^i \mathbf{s} \tag{3}$$

$$w(i) = e^{-\xi i} \tag{4}$$

Where $\mathbf{A}$ is the adjacency matrix of the direct product of the two graphs, $w(i)$ is the relative weight of $i$ long walks, $N$ is the maximum length of the walks considered, while $\mathbf{s}$ and $\mathbf{e}$ are vectors containing the probabilities of starting and ending the walks on the given node respectively. These vectors might be determined using a priori information on the graphs, or using the node weights in the case of node weighted graphs. Also $\xi$ is hyperparameter controlling the slope of the weights of walks of different lengths.

It is important to node, that by computing the direct product of graphs the random walk kernel is capable of comparing graphs with different number of nodes. Furthermore, since the random walk kernel computes the weighted scalar product of the node vectors, it is also invariant to node ordering.

## 3. 3D Object Recognition

In this section we present our 3D shape recognition algorithm. In the first subsection we explain our shape representation method, while in the second subsection we describe the learning classification algorithm. In the third part of this section we will present our method for determining the optimal hyperparameters of ht classification algorithm, while in the last subsection we will present the localization algorithm.

### 3.1. Shape Description

Computing the representation of the shape of 3D scenes is the first part of our algorithm. Here, we used a method that describes the geometry of an entire scene, not just an object, still it allows as to extract smaller parts of the given scene and examine them individually. The logic behind this is that we wanted to avoid using a segmentation method to create object candidates, since this arbitrary step would greatly influence the results of the detection algorithm.

Our method is similar to the one used by Schnabel et. al.[11] Instead of segmenting the scene into object candidates, we break it down into primitive shapes, which can be interpreted

as basic building blocks of the scene's geometry. By doing this we implicitly assume that certain objects can be represented as combinations of primitive shape. This way the boundaries between objects can be determined by a learning algorithm by assigning primitive shapes to an object category.

To implement the segmentation step we use the efficient RANSAC algorithm developed by Schnabel et. al.[12] Using this algorithm we are able to break down the 3D scene into the following five primitive shapes: spheres, cones, cylinders, planes and tori (the primitives used in their implementation), and construct a graph from them. (**Figure 1**)

However, instead of the brute force graph matching method they used to detect objects, we will use a learning algorithm. There are several reasons behind this. First, our data is noisy, and the 3D reconstruction of objects may not be complete (there are no images available for all viewpoints), and some objects cannot be accurately represented with the primitive shapes listed above, all of which will cause the segmentation algorithm to produce incorrect results. On the other hand, our aim is to recognize object *categories*, therefore we will have to account for intra-class variance.

Furthermore, instead of applying constraints on the nodes or edges of the constructed graph, which machine learning algorithms handle poorly, we introduce features for both nodes and edges that will aid the algorithm to find similar structures within graphs. For the nodes of the graph - which represent primitives - we compute features using the point cloud of inliers determined by the RANSAC algorithm. Since there are five primitive shape categories with different possible features (shown in the table below) we would need to construct separate feature vectors for each primitive type, which would make comparing nodes difficult. In order to avoid this we construct just one feature vector for nodes by concatenating the feature vectors for the different primitive types. The features that belong to another primitive type are set to zero.

| | |
|---|---|
| Plane | Area<br>Diameter<br>Bounding Box Area |
| Sphere | Radius |
| Cone | Radius<br>Height |
| Cylinder | Radius<br>Height<br>Angle |
| Torus | Inner Radius<br>Body Radius |

It is important to note, that it is possible to assign a coordinate system to all primitive shapes that can be used to describe geometric relations between given nodes. However, almost all shapes have symmetric properties, therefore these coordinate systems will be ambiguous. Nevertheless, we can still assign an origin to all shapes unambiguously, and a direction to all primitives, with the exception of the sphere. The origins and the directions are shown for each primitive type in the table below.

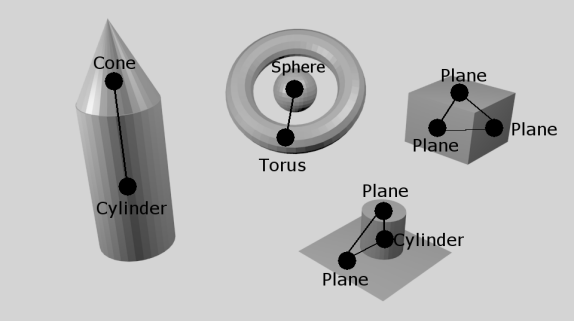


**Figure 1:** *The graph constructed from primitive shapes (only significant edges are drawn)*

| Primitive | Center | Direction |
|---|---|---|
| Plane | Centroid | Normal vector |
| Sphere | Centroid | z direction |
| Cone | Peak | Axis |
| Cylinder | Middle of the axis | Axis |
| Torus | Center | Axis |

The edges of the graph represent the relations of these coordinate systems. We store the translation and the rotation between the two coordinate systems. Because of the ambiguity or these coordinate systems, we determine the smallest possible rotation between the two vectors representing the directions. Also, since the world coordinate system might be different for different scenes, we use only the magnitude of translation and the angle of rotation between to coordinate systems as features for the learning algorithm.

Another essential property of our shape description algorithm is that we always create full graphs. This way we need no arbitrary threshold to determine the adjacency of nodes, since this property has already been encoded in the edge features. Moreover, this way the adjacency property remains a real value instead of a binary one, which allows for a more robust solution.

### 3.2. Learning Graph Classification

The core of our object recognition algorithm is a learning graph classification algorithm. Here, we use a simple SVM that uses the Random Walk kernel, since this way we can achieve complete invariance to node ordering. However, the

standard Random Walk kernel takes simple weighted graphs as input, therefore we need to use the extension of the kernel for graphs with vectorial node and edge weights. In order to achieve that, we define vectorial kernel functions between node and edge feature vectors, and use these kernels to calculate the node vector and the adjacency graph of the direct product graph.

$$\mathbf{N}_{in'+j} = K_N(n_i, n'_j) \tag{5}$$

$$\mathbf{A}_{(in'+j),(jn+i)} = K_E(e_{i,j}, e'_{j,i}) \tag{6}$$

Where **N** and **A** are the node vector and the adjacency matrix of the direct product graph, *n, n', e, e'* are the nodes and edges of the two graphs compared, while $K_N$ and $K_E$ are the node and edge kernels respectively.

We will use the node kernel to construct a vector of node weights of the direct product graph, where nodes with high weights belong to nodes that were similar to each other in the original graphs. We will use the node weight vector as the starting and ending probability vector of the random walk kernel. This choice means that we are more likely to start and end walks on nodes that look similar. Also, we will not normalize the node vector (though the probabilistic interpretation of the random walk kernel would require it), since the length of the node vector contains important information on the similarity of the graphs in itself.

There is one more difficulty in creating the node kernel, namely that many features we calculated for certain primitive shapes differ significantly in scale. There are area, length and angle features, and if we simply compared the feature vectors of two nodes, the differences between area-type features would determine the error value. In order to avoid this we create a normalized error function between node vectors, and then use a standard RBF kernel to implement the node kernel.

$$E = \sum_{i=1}^{M} \frac{|f_i - f'_i|}{f_i + f'_i} \tag{7}$$

$$K_N = e^{-\theta \cdot E} \tag{8}$$

Where $f_i$ , $f'_i$ are the $i_{th}$ feature of the nodes $n$ and $n'$, and $\theta$ is a hyperparameter that controls the slope of the RBF kernel.

Similarly to the node kernel, we will use a kernel function between edges to create the adjacency matrix of the direct product graph that is needed for computing the random walk kernel. Intuitively, the weight of an edge corresponds to the probability of taking that edge during a random walk, therefore the edge kernel must be designed to take that interpretation into account.

Thus we construct an edge kernel that assigns higher weight to edges that are similar, but also to edges that are between close nodes. This way, we are more likely to walk between close nodes during the random walks, therefore we will explore the local structure of the graph more thoroughly. Moreover, by introducing this in the edge kernel we accounted for the adjacency of the nodes of graphs, however our method is much more flexible that a fix adjacency criterion. to calculate the kernel value, we use an RBF function here as well.

$$K_E = e^{(-\gamma \cdot \Delta\phi - \delta \cdot \Delta d - \mu \cdot d)} \tag{9}$$

Where $\Delta\phi$ and $\Delta d$ denote the difference in the angles and lengths of the two edges compared, $d$ is the average of the distance of the two edges, while $\gamma$, $\delta$, and $\mu$ are hyperparameters. It is important to note that the random walk kernel introduces two further hyperparameters: $N$, which is the maximum length of the random walks considered in the kernel, and $\xi$, which is the slope of the weights of random walks of different lengths.

### 3.3. Optimizing Kernel Parameters

In the previous section we have introduced our application of the random walk kernel for the problem of shape recognition. There are several hyperparameters of the kernel function that also need to be optimized in order to create the learning algorithm. However, because of the inherent complexity of the random walk kernel these parameters are difficult to optimize through analytical or numerical means, therefore finding their optimal value is a different task altogether.

In order to solve this problem we employ a genetic algorithm that will be used to approximate the optimal values of the hyperparameters by maximizing the cross-validation accuracy of the classification algorithm. The list of hyperparameters we optimize is presented in the table below.

| | |
|---|---|
| $C$ | The C parameter of SVM |
| $N$ | Maximum length of walks |
| $\xi$ | The slope of the relative weights of walks of different lengths |
| $\gamma$ | The relative weight of the angle difference between edges |
| $\phi$ | The relative weight of the distance difference between edges |
| $\mu$ | The weight punishing edges of large distances |
| $\theta$ | The parameter of the node kernel |

For the optimization we use a simple genetic algorithm

with stochastic universal selection and no generation overlap. Parameters are represented as real numbers in the genome with a real mutation operator. The fitness score of the individuals is produced from cross-validation accuracy that is achieved through a 10-fold validation method.

Since the size of the population is usually large, the evaluation of the random walk kernel function becomes a significant bottleneck of the algorithm. Since the kernel needs to be evaluated for the same training data several times with different hyperparameters, parallelization is a straightforward way to reduce computation time. In order to speed up the optimization method, we created a GPGPU implementation of the random walk kernel in CUDA.

### 3.4. Localization

The final part of our learning 3D object recognition algorithm is the localization method that employs the classification algorithm to extract subgraphs corresponding to the given object categories. To achieve this, we use a graph building algorithm based on the result of the classification. It is an iterative algorithm that selects a starting node and adds new nodes to the graph in each iteration, trying to increase the classification probability of the new graph. Therefore, it uses a binary classification algorithm that assigns the probability of the labels to each object instead of just the labels themselves.

The first step of the graph building algorithm is starting node selection. Here, the classification probability could be used, however, since an average object consists of 3-5 nodes, just one node might achieve a very low score regardless of whether it is part of the correct object or not. Therefore, we assign a score to nodes using a modification of the random walk kernel that assigns the number of simultaneous random walks starting on the given node to each of the nodes. Since the random walk kernel assigns this score to the nodes of the direct product graph, we need to sum the values corresponding to the same node in the scene graph. Here, the scene graph is matched against the most similar support vector graph.

$$N_i = \sum_{i=in'}^{in'+n'-1} \sum_{k=1}^{N} \mathbf{A}^k \mathbf{s} \tag{10}$$

Where $\mathbf{A}$ is the adjacency matrix of the direct product graph, $\mathbf{s}$ is the vector of starting probabilities, and $n'$ is the number of nodes in the support vector graph used for comparing. Once the starting node has been selected, the localization method will try to add further nodes and the corresponding edges to this starting graph, so that it would increase the classification probability. It is worth mentioning, that since objects are local phenomena, the algorithm only considers nodes that are close to the graph (the smallest distance between the node and the nodes of the graph). The algorithm is illustrated and described in **Algorithm 1**, and **Figure 2**.

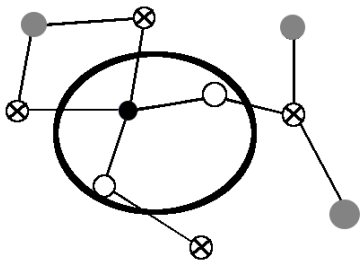

**Figure 2:** *Localization: The starting node is represented by the filled black dot, and the close nodes (unfilled) are added to the graph. The ones that do not increase the classification score (filled with an x) are then removed. The gray nodes are not considered because of their large distance from the rest of the nodes.*

**Algorithm 1** Localization using Classification

$\{G\} = \emptyset$
$\forall n \rightarrow goodness(n)$
**while** $\exists n : goodness(n) > Th_g$ **do**
  $G = n$
  **while** $\exists n : ||G - n||^2 < Th_d$ **do**
    $G' \leftarrow G + n$
    **if** $C_p(G') > C_p(G)$ **then**
      $G = G'$
    **end if**
  **end while**
  **if** $C(G) = 1$ **then**
    $\{G\} \leftarrow G$
  **else**
    stop
  **end if**
**end while**

In order to create object candidates for all object categories the localization algorithm is performed independently for all categories. However, this creates the possibility of overlapping objects: objects from different categories containing the same nodes, therefore occupying the same 3D space. To resolve the conflicts between object candidates we use a post-processing step to select a subset of non-overlapping candidates that maximize the sum of classification probability.

The last step of the localization method is pose estimation. Here, we match the nodes of the object candidate graph an the most similar positive support vector graph, and perform a

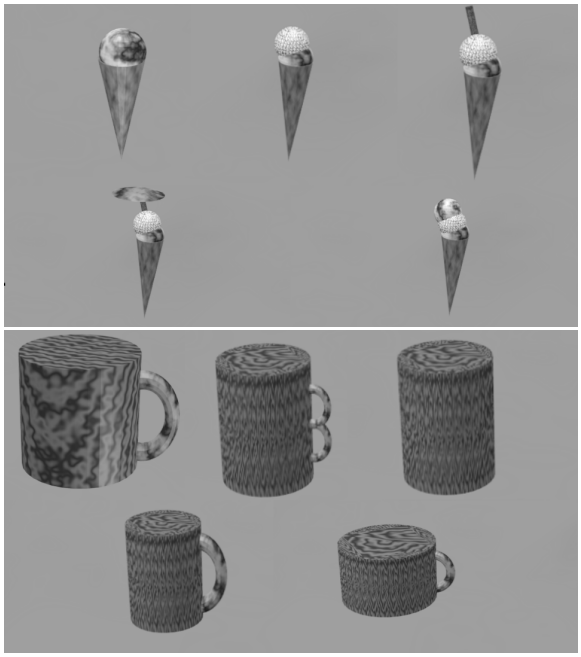

**Figure 3:** *The ice cream and mug categories. All five variations shown.*

least-squares pose estimation using the corresponding node centroids and directions.

## 4. Results

In this section we will present the methods for testing the algorithms described in the previous section, and the results of the tests. In order to test the learning and localization algorithms we require multiple images of scenes with a known labeling of the objects within the scene. We use the 3D reconstruction and segmentation algorithms to construct training graphs for the given object categories according to the labeling. The accuracy of the localization algorithm will then be determined by the number of graph nodes assigned to the correct category.

We used Blender (a 3D modeling application) to create controlled test scenes with five different object categories (mug, house, ice cream, snowman and column) in them. We took hundreds of images of each object from different angles, with different lighting conditions and in-class variation (**Figure 3**). The scenes used for training have only one instance of one object category present, while the test scenes (**Figure 4**) have at least one instance of all five categories (on some images occlusion is possible). Training and test graphs are produced by using a structure from motion reconstruction algorithm on subsets of the images taken of the same object under static lighting conditions.

We then labeled the training graphs based on the label of the image that the graph was created from, and trained binary classifiers for each object category against all the remaining object categories.Cross-validation is performed only once, since we assume that the optimal hyperparameters for one category will be close to the optimum for other categories as well. The training and validation accuracies of the classification algorithm and the optimal hyperparameters are shown in **Tables 1** and **2** accordingly.

| Category | Training | Validation | Localization |
|---|---|---|---|
| Mug | 4.05% | 7.15% | 9.85% |
| Column | 1.34% | 3.97% | 8.85% |
| Snowman | 4.28% | 5.98% | 6.46% |
| House | 2.48% | 4.02% | 7.36% |
| Ice cream | 5.25% | 8.41% | 8.15% |

**Table 1:** *The Error Rate of the Algorithm*

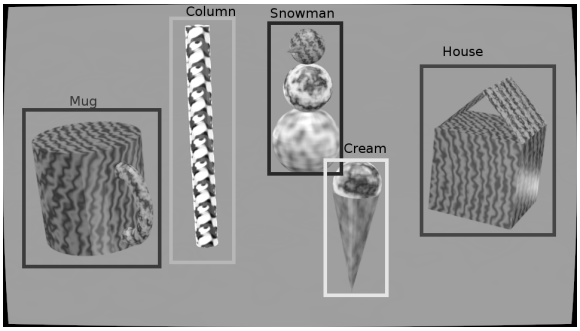


**Figure 4:** *The test scene with all categories present. Labeling is also displayed.*

| $\theta$ | $\gamma$ | $\delta$ | $\mu$ | $C$ | $N$ | $\xi$ |
|---|---|---|---|---|---|---|
| 0.638 | 0.527 | 0.355 | 0.023 | 1.87 | 9 | 12.6 |

**Table 2:** *Determined Parameters*

The localization algorithm is evaluated on test images with all five categories present. These images are only labeled by bounding boxes, therefore we cannot predict the position and orientation of the objects reliably. Because of that, the accuracy of the localization algorithm will be represented by the number of nodes labeled correctly by the algorithm. The results of the localization method are shown in **Table 1**.

## 5. Conclusion

In this paper we presented an object recognition algorithm that uses a graph-based shape representation to perform matching based on form similarity. To achieve that, we used a graph kernel-based SVM classification algorithm and a localization method. According to our test results both of these algorithms are capable of producing satisfying results.

Our major goals for future research include replacing the current localization algorithm with graph matching. We will use a semi-supervised method to learn reference graphs for object categories. The algorithm will compute descriptor vectors for every node in the graphs and perform clustering in order to create reference graphs. This method can be extended further by using differences between the real scene and 3D models of the virtual objects put into the scene to compute a fit error that may be used to label scenes automatically.

## References


1. Hiroshi Ishii, Brygg Ullmer, Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms, *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, pp 234–241, 1997.
2. Ivan Poupyrev , Desney S Tan , Mark Billinghurst , Hirokazu Kato , Holger Regenbrecht , Nobuji Tetsutani , Daimlerchrysler Ag, Tiles: A Mixed Reality Authoring Interface, 2001. *Proceedings of IFIP INTERACT01: Human-Computer Interaction*, pp 334-341.
3. Mark Billinghurst, Hirokazu Kato, Seiko Myojin, Advanced Interaction Techniques for Augmented Reality Applications, *Third International Conference on Virtual and Mixed Reality*, pp 13–22, 2009.
4. Deepak Bandyopadhyay, Ramesh Raskar, Henry Fuchs, Dynamic Shader Lamps : Painting on Movable Objects, *The Second IEEE and ACM International Symposium on Augmented Reality*, pp 207–216, 2001.
5. Kresimir Matkovic, Thomas Psik, Ina Wagner, Denis Gracanin, Dynamic Texturing of Real Objects in an Augmented Reality System *Proceedings of IEEE Virtual Reality 2005*, pp 245–248, 2005.
6. Wolfgang Broll, Eckhard Meier, Thomas Schardt, The Virtual Round Table - a Collaborative Augmented Multi-User Environment, *Proc. of the ACM Collaborative Virtual Environments*, pp 39–46, 2000.
7. Robert Osada, Thomas Funkhouser, Bernard Chazelle, David Dobkin, Matching 3D Models with Shape Distributions, *International Conference on Shape Modeling and Applications*, pp. 154–166, 2001.
8. Martin A. Fischler, Robert C. Bolles, Random Sample Consensus: A Paradigm for Model Fitting with Apphcatlons to Image Analysis and Automated Cartography, *ACM Graphics and Image Processing*, **24**(6):381–395, 1981.
9. D.H. Ballard, Generalizing the Hough transform to detect arbitrary shapes, *Pattern Recognition*, **13**26):111–122, 1981.
10. Aleksey Golovinskiy, Vladimir G. Kim, Thomas Funkhouser, Shape-based Recognition of 3D Point Clouds in Urban Environments, *12th IEEE International Conference on Computer Vision*, pp 2154-2161 2009.
11. R. Schnabel, R. Wahl, R. Wessel, R. Klein, Shape Recognition in 3D Point Clouds, *The 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*,**8**, pp 65–72, 2008.
12. Ruwen Schnabel, Roland Wahl, Reinhard Klein, Efficient RANSAC for Point-Cloud Shape Detection, *Computer Graphics Forum*, **26**(2):214–226, 2007.
13. F. Chung, Spectral graph theory, *American Mathematical Society*, 1997.
14. David H. White, Mixing spectral representations of graphs, *18th International Conference on Pattern Recognition*, **4**, pp 140–144, 2006.
15. S.V.N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, Karsten M. Borgwardt, Graph Kernels, *Journal of Machine Learning Research*, **11**, pp 1201–1242, 2010.